Homework 5

PSTAT 131/231

Contents

Elastic Net Tuning

For this assignment, we will be working with the file `"pokemon.csv"`, found in `/data`. The file is from Kaggle: https://www.kaggle.com/abcsds/pokemon.

The Pokémon franchise encompasses video games, TV shows, movies, books, and a card game. This data set was drawn from the video game series and contains statistics about 721 Pokémon, or "pocket monsters." In Pokémon games, the user plays as a trainer who collects, trades, and battles Pokémon to (a) collect all the Pokémon and (b) become the champion Pokémon trainer.

Each Pokémon has a primary type (some even have secondary types). Based on their type, a Pokémon is strong against some types, and vulnerable to others. (Think rock, paper, scissors.) A Fire-type Pokémon, for example, is vulnerable to Water-type Pokémon, but strong against Grass-type.

Figure 1: Fig 1. Vulpix, a Fire-type fox Pokémon from Generation 1.

The goal of this assignment is to build a statistical learning model that can predict the **primary type** of a Pokémon based on its generation, legendary status, and six battle statistics.

Read in the file and familiarize yourself with the variables using `pokemon_codebook.txt`.

```
library(tidymodels)
library(ISLR)
library(ISLR2)
library(tidyverse)
library(glmnet)
tidymodels_prefer()
```

Note: Most codes are basically from lab 5.

Exercise 1

Install and load the `janitor` package. Use its `clean_names()` function on the Pokémon data, and save the results to work with for the rest of the assignment. What happened to the data? Why do you think `clean_names()` is useful?

```
library(janitor)
Pokemon <- read_csv("Pokemon.csv")
Pokemon <- clean_names(Pokemon)
```

Answer: From the description. By clean_names(), the resulting names are unique and consist only of the underscore, numbers, and letters, and by default in snake case. It makes calling variables easier and gets rid of unreadable characters.

Exercise 2

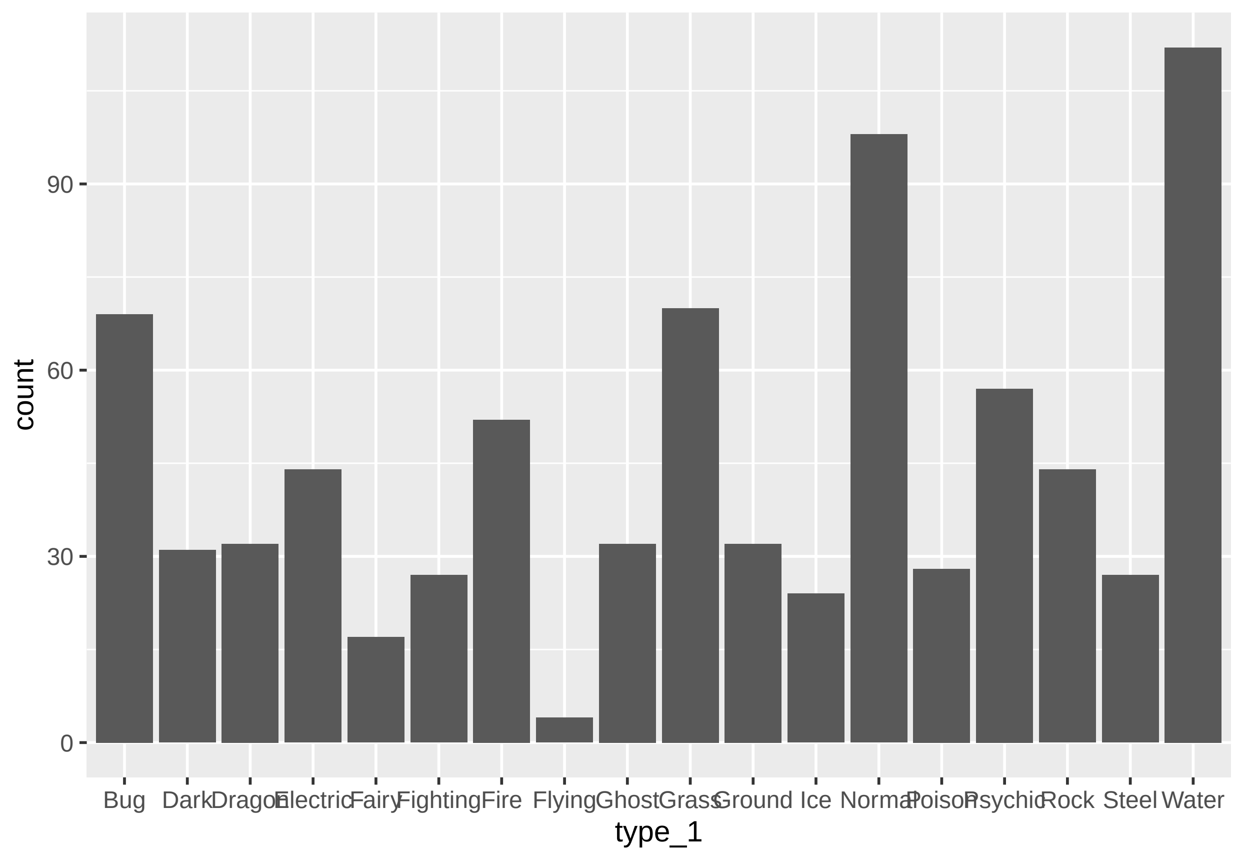
Using the entire data set, create a bar chart of the outcome variable, `type_1`.

How many classes of the outcome are there? Are there any Pokémon types with very few Pokémon? If so, which ones?

For this assignment, we'll handle the rarer classes by simply filtering them out. Filter the entire data set to contain only Pokémon whose `type_1` is Bug, Fire, Grass, Normal, Water, or Psychic.

After filtering, convert `type_1` and `legendary` to factors.

```
p <- ggplot(data=Pokemon, aes(x=type_1)) +
     geom_bar(stat = "count")
p
```



```
Pokemon <- Pokemon[Pokemon$type_1 %in% c('Bug', 'Fire', 'Grass', 'Normal', 'Water', 'Psychic'), ]
Pokemon$type_1 = factor(Pokemon$type_1)
Pokemon$legendary = factor(Pokemon$legendary)
Pokemon$generation = factor(Pokemon$generation) # When model fitting, it says generation is not factor
```

Answer: There are 18 classes of outcome. There are very few in the flying class (and the fairy class but it's not as obvious as flying).

Exercise 3

Perform an initial split of the data. Stratify by the outcome variable. You can choose a proportion to use. Verify that your training and test sets have the desired number of observations.

```
set.seed(1203)
Pokemon_split <- initial_split(Pokemon, strata = "type_1", prop = 0.68)

Pokemon_train <- training(Pokemon_split)
Pokemon_test <- testing(Pokemon_split)

dim(Pokemon_train)
```

```
## [1] 308  13
```

```
dim(Pokemon_test)
```

```
## [1] 150  13
```

Answer: The training and test sets have the desired number of observations.

Next, use *v*-fold cross-validation on the training set. Use 5 folds. Stratify the folds by `type_1` as well. *Hint: Look for a `strata` argument.* Why might stratifying the folds be useful?

```
Pokemon_5fold <- vfold_cv(Pokemon_train, v = 5, strata = type_1)
```

Answer: Stratifying the folds can be useful because it keeps the distribution (proportion of variable types) in each fold to be the same.

Exercise 4

Set up a recipe to predict `type_1` with `legendary`, `generation`, `sp_atk`, `attack`, `speed`, `defense`, `hp`, and `sp_def`.

- Dummy-code `legendary` and `generation`;
- Center and scale all predictors.

```
recipe <-
  recipe(formula = type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_def, d
  step_dummy(c('legendary', 'generation')) %>%
  step_normalize(all_predictors())
```

Exercise 5

We'll be fitting and tuning an elastic net, tuning `penalty` and `mixture` (use `multinom_reg` with the `glmnet` engine).

Set up this model and workflow. Create a regular grid for `penalty` and `mixture` with 10 levels each; `mixture` should range from 0 to 1. For this assignment, we'll let `penalty` range from -5 to 5 (it's log-scaled).

How many total models will you be fitting when you fit these models to your folded data?

```
enet <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

workflow <- workflow(recipe, enet)

penalty_mixture_grid <- grid_regular(penalty(range = c(-5, 5)), mixture(range = c(0, 1)), levels = 10)
```

Answer: Because the grid has 10 level penalty and 10 level mixture, and we set 5 folds, so there are 10x10x5 = 500 models to be fitted in total.

Exercise 6

Fit the models to your folded data using `tune_grid()`.

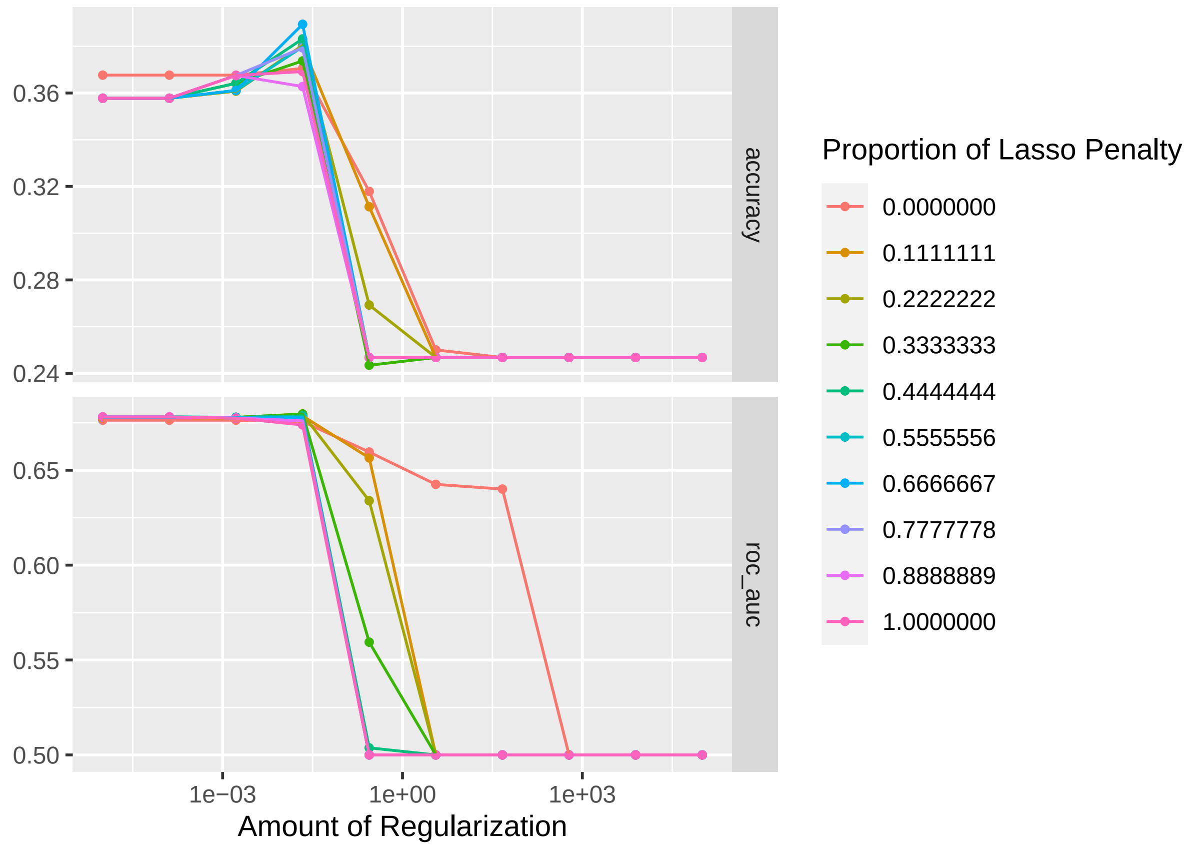
```r
tune_res <- tune_grid(
  workflow,
  resamples = Pokemon_5fold,
  grid = penalty_mixture_grid
)

save(tune_res, file = "HW5_fit.rda")
```

load saved models

```r
load(file = "HW5_fit.rda")
```

Use `autoplot()` on the results. What do you notice? Do larger or smaller values of `penalty` and `mixture` produce better accuracy and ROC AUC?

```r
autoplot(tune_res)
```



Answer: Larger values of `penalty` and `mixture` produce lower accuracy and ROC AUC.

Exercise 7

Use `select_best()` to choose the model that has the optimal `roc_auc`. Then use `finalize_workflow()`, `fit()`, and `augment()` to fit the model to the training set and evaluate its performance on the testing set.

```
best_penalty <- select_best(tune_res, metric = "roc_auc")
enet_final <- finalize_workflow(workflow, best_penalty)
enet_final_fit <- fit(enet_final, data = Pokemon_train)

enet_final_fit_on_test <- augment(enet_final_fit, new_data = Pokemon_test)
```

Exercise 8

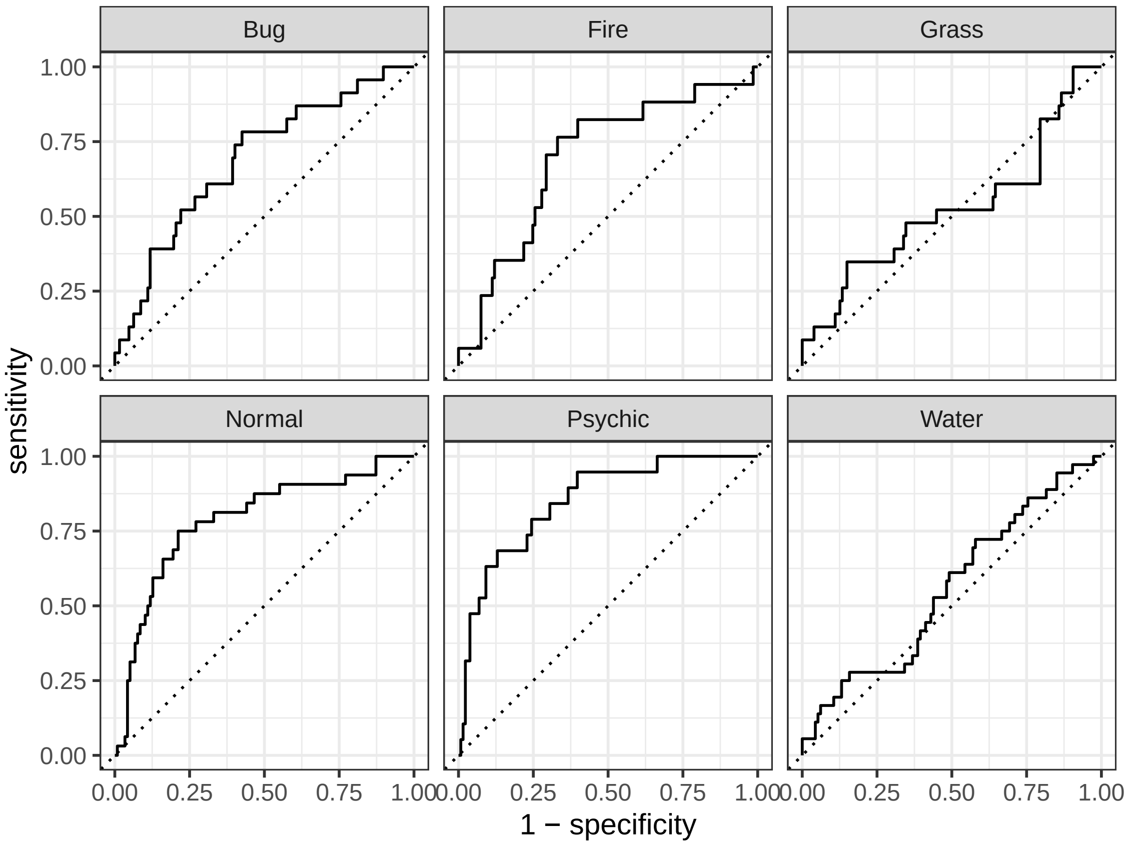
Calculate the overall ROC AUC on the testing set.

```
ROCAUC <- roc_auc(data = enet_final_fit_on_test, truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .p
ROCAUC
```

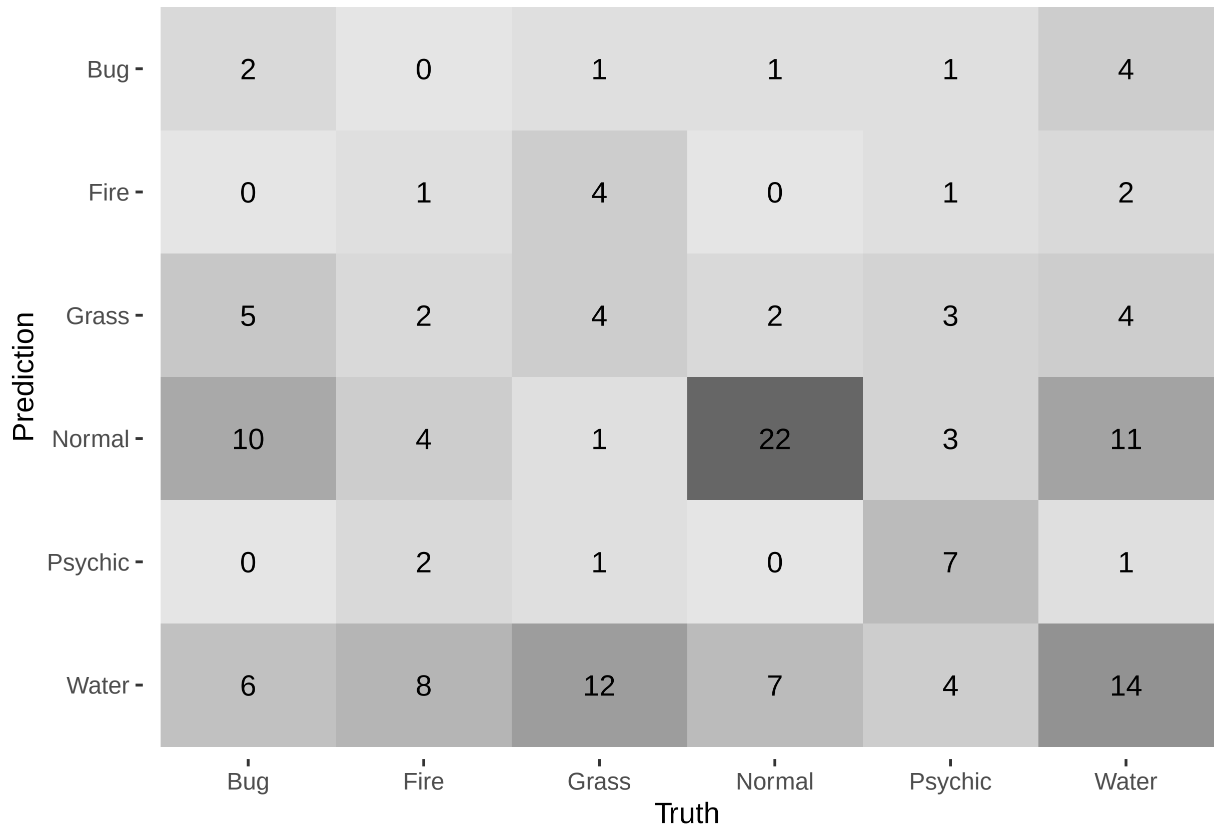
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.682
```

Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

```
ROC_Curve <- roc_curve(data = enet_final_fit_on_test, truth = type_1, estimate = c(.pred_Bug, .pred_Fir
autoplot(ROC_Curve)
```



```r
CONF_MAT <- conf_mat(data = enet_final_fit_on_test, truth = type_1, estimate = .pred_class)
autoplot(CONF_MAT, type = "heatmap")
```



What do you notice? How did your model do? Which Pokemon types is the model best at predicting, and which is it worst at? Do you have any ideas why this might be?

Answer: The ROC curves indicates that this model performs poorly in predicting grass and water classes. And from the heat map, the model doesn't do well at all besides predicting the normal class. I thought it was because of the unbalanced distribution such that Normal has more data to train from. However, it should be false because we used stratified sampling to maintain the same distribution. And there are more observations in the water class but it performs much worse than the normal class, so the class size isn't a valid reason. In my guess, the independent variables in fact have little correlations to the response variable and so we can't create a logical prediction between them.

For 231 Students

Exercise 9

In the 2020-2021 season, Stephen Curry, an NBA basketball player, made 337 out of 801 three point shot attempts (42.1%). Use bootstrap resampling on a sequence of 337 1's (makes) and 464 0's (misses). For each bootstrap sample, compute and save the sample mean (e.g. bootstrap FG% for the player). Use 1000 bootstrap samples to plot a histogram of those values. Compute the 99% bootstrap confidence interval for Stephen Curry's "true" end-of-season FG% using the quantile function in R. Print the endpoints of this interval.